Software Architecture

Introduction To Documenting Software Architectures

Duc-Man Nguyen, Ph.D
mannd@duytan.edu.vn
0904 235 945

1

Outline

- Role of documentation in the process
- Qualities of good documentation
- View based documentation
- View organization
- Designing the documentation

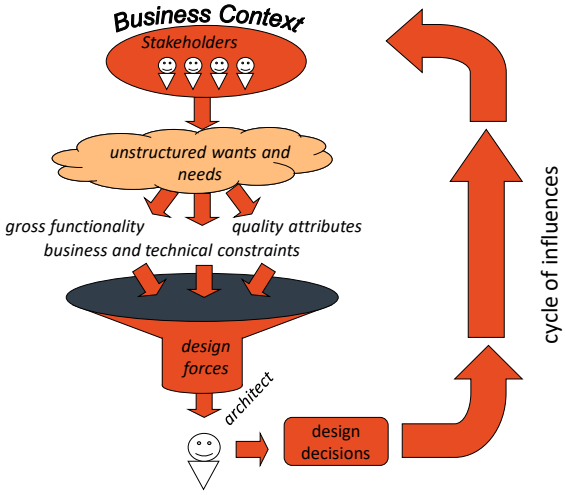
2

Outline

- Role of documentation in the process
- Qualities of good documentation
- View based documentation
- View organization
- Designing the documentation

3

Recall …



4

What Would This Allow Us To Do? - 1

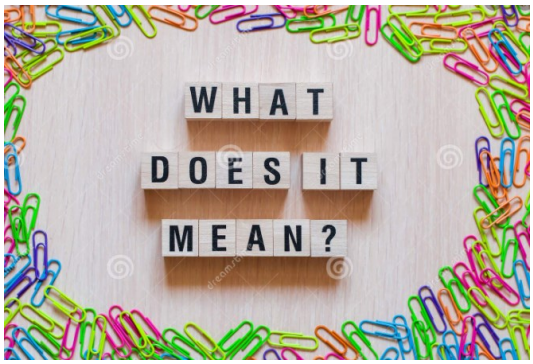
- First we this enables alignment between the business context and the resulting system
  - Thus the system will support the strategic direction of the organization
- It also allows for strategic evolution of the system
  - We could predict the properties that would emerge from evolutionary changes
  - We could maintain the alignment between the business context and the system over time

What Would This Allow Us To Do? - 2

- Strategically determine when to sunset the system
  - When the effort to support the desired evolution of the organizational strategy becomes too great we can plan to sunset the system

What Does This Mean? - 1

- We've already said (many times) that we need to maintain traceability from the business context to the architectural decisions



What Does This Mean? - 2

- What does this mean?
  - We need to understand the specific relationship between the business context and the architectural drivers
  - We need to understand the specific relationship between the architectural drivers and the architectural decisions
  - We need to ensure that the implementation realizes the architectural decisions
  - We need to understand the impact of changes in the architecture

How Do We Do This?

- It isn't reasonable that we keep track of all of these things in our head
  - I can't even remember what I did yesterday
- We need to have some kind of documentation that supports these needs (and others yet to be mentioned)
- In this section we are going to look in more detail at the needs that documentation fulfills
- We will also look at a strategy for designing and creating software architectural documentation

Software Architecture → Communication Vehicle - 1

- We've said that the software architecture is a vehicle for communication
- We've also said that a software architecture is an abstraction
  - Recall that an abstraction promotes some details and omits others
  - A good abstraction promotes only those details needed for a given purpose

Software Architecture → Communication Vehicle - 2

- Keep both of these things in mind as we talk about documentation
  - We have to explicitly understand what needs to be communicated
  - We also need to know what details are needed to support the communication intent

Reader's Perspective - 1

- Another way to say this is that the documentation needs to be written from the reader's perspective
- This is rarely done
  - More often we write documentation from the writer's perspective
  - We attempt to capture what is in our head (not all of which is relevant for any given purpose)
  - We don't organize the relevant information explicitly to support the needs

Reader's Perspective - 2

- High maturity organizations often have a template which guides documentation
  - This template is often designed to ***generally*** support the reader's needs, but does not explicitly support specific needs for a specific project

Outline

- Role of documentation in the process
- Qualities of good documentation
- View based documentation
- View organization
- Designing the documentation

General Qualities

- We will talk about how to design documents to support specific needs in a minute, but first let's look at some general qualities for good documentation

Seven Principles of Documentation

1. Write from the point of view of the stakeholder.
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record rationale.
6. Keep documentation current but not too current.
7. Review documentation for fitness of purpose.

Write From The Stakeholder's Perspective

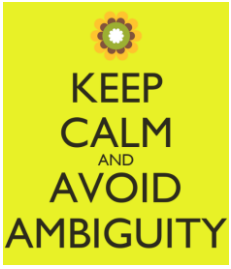
- What will the reader want to know when reading a document?
  - Make information easy to find!
  - Your reader will appreciate your effort and be more likely to read your document.
- Avoid writing for your (i.e. the writer's) convenience.
  - stream of consciousness: the order is that in which things occurred to the writer
  - stream of execution: the order is that in which things occur in the computer during program or task execution

Avoid Unnecessary Repetition

- Each kind of information should be recorded in exactly one place.
  - This makes documents easier to use and easier to change and more likely that they will be maintained over the lifetime of the system.
  - Repetition often confuses, because the information is repeated in slightly different ways.

Avoid Ambiguity - 1

- Architecture documentation is a communications vehicle. If the reader misunderstands, the documentation has failed.
- Box-and-line diagrams are a common form of architectural notation, but what do they mean?



Avoid Ambiguity - 2

- Always include a key or legend.
- If a common language and/or notation is used, point to the formal definition – don't assume everyone knows the notation.
- Give the meaning of each symbol and each line.
- Remain consistent in the use of the symbols.

Use a Standard Organization

- Establish it, make sure your writers adhere to it, and make sure that readers know what it is.
  - helps the reader navigate and find information
  - helps the writer place information and measure work left to be done
  - embodies completeness rules, and helps writers check for completeness as the write
- Organize the documentation for the reader's ease of reference not for the convenience of the writer.
  - A successful document will be *referred to* many times.

Record Rationale

- Why did you make certain design decisions?
  - Next week, next year, or next decade, how will you remember? How will the next designer know?
  - Recording rationale requires a cultivated discipline, but saves enormous time in the long run.
  - Record rejected alternatives and reasons for rejection as well.
  - Make it a habit to carry an engineering notebook with you throughout the project.

Keep Documentation Current…

- Documentation that is incomplete, out of date, does not reflect truth.
- Documentation that is kept current is used.
- With current documentation, questions are most efficiently answered by referring the questioner to the documentation.
- If a question cannot be answered with a document, fix the document and then refer the questioner to it.
- This sends a powerful message to readers.

…But Not Too Current

- During the design process, decisions are considered and re-considered with great frequency.
- Revising the documentation every five minutes will result in unnecessary expense, inertia, and resistance to change due to “paperwork pain.”
- Choose points in the development plan when documentation is brought up to date – schedule it like you would any other task.
- Follow a release strategy that makes sense for your project.

Review Documentation For Fitness of Purpose

- Only the intended users of a document can tell you if it
  - contains the right information
  - presents the information in a useful way
  - satisfies their needs
- Plan to review your documents with stakeholders for whom it was created.

Outline

- Role of documentation in the process
- Qualities of good documentation
- View based documentation
- View organization
- Designing the documentation

Architectural Views – 1

- An architecture is a very complicated construct and its almost always too complicated to be seen all at once.
- Software systems have many structures or views.
  - Just as buildings have drawings describing electrical systems, plumbing, structure, so it is with software.
  - No single representation structure or artifact can be *the* architecture.
  - The set of candidate structures is not fixed or prescribed: *architects need to select what is useful for analysis or communication*.

Architectural Views – 2

- Systems are composed of many structures
  - components and connectors
  - modules, showing composition/decomposition, mapping to code units
  - processes, and how they synchronize
  - programs, and how they call or send data to each other
  - how software is deployed on hardware
  - how teams cooperate to build the system

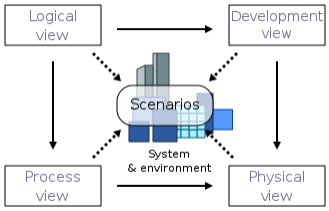
Architectural Views – 3

- A view is a representation of a set of system elements and the relations associated with them.
  - Not *all* system elements -- *some* of them.
- A view binds an element *type* and relation *type* of interest, and shows those.
- In box-and-line diagrams, another way of asking what the boxes and lines mean is:
  - What *element types* and *relation types* are you showing?
  - In other words, "What *view* are you showing?"

What Kind of Views Are There? – 1

•Kruchten's 4+1 views*:

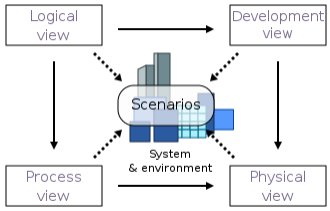
- Logical view: supports behavioral requirements. Key abstractions, which are objects or object classes
- Process view: addresses concurrency and distribution. Maps threads to objects.



*Kruchten, P. The 4+1 view Model of Software Architecture. *IEEE Software*, vol. 12, no. 6, November 1995, pp. 42-50.

What Kind of Views Are There? – 2

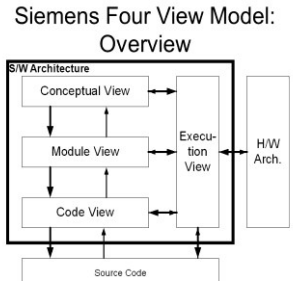
- Development view: organization of software modules, libraries, subsystems, units of development.
- Physical view: maps other elements onto processing and communication nodes.
- "Plus one" view: Maps the other views onto important use cases to show how they work together.



What Kind of Views Are There? – 3

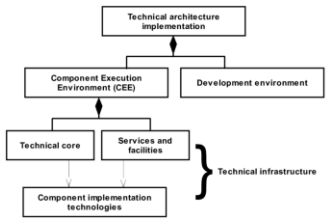
- Siemens Four-Views*
  - conceptual view – elements and their relationships
  - module interconnection view – functional decomposition
  - execution view – dynamic structures
  - code view – source code organization

*Hofmeister C.; Nord R.; Soni D.; *Applied Software Architecture* Reading, MA: Addison-Wesley, 1999



What Kind of Views Are There? – 3

- Herzum & Sims*
  - technical architecture – technical services and facilities
  - application architecture – patterns, guidelines, standards
  - project management architecture – management tools
  - functional architecture – specification, implementation



*Herzum P.; Sims O.; *Business Component Factory*, New York, NY: John Wiley and Sons,1999

What Kind of Views Are There? – 4

- Software Cost Reduction Method*
  - module view - shows modules as units of encapsulation; used to isolate changes and achieve modifiability
  - process view - shows processes and how they synchronize and communicate at run-time; used to achieve performance
  - uses view - shows programs and how they depend on each other; used to achieve incremental development and the ability to quickly field subsets

*Clements P., Parnas D., Weiss D., "The Modular Structure of Complex Systems", Proceedings, Seventh International Conference on Software Engineering, pp. 408-417, Mar. 1984. Reprinted in IEEE Transactions on Software Engineering, vol. SE-11, pp. 259-266, March 1985. Reprinted in Software Fundamentals: Collected Papers by David L. Parnas, ed. by D. Hoffman and D. Weiss, Addison Wesley Longman, 2001.

Which Views are Relevant?

- Which views are relevant? It depends on
  - who the stakeholders are
  - how they will use the documentation.
- Three primary uses for architecture documentation
  - Education - introducing people to the project.
  - Communication - among stakeholders.
  - Analysis - assuring quality attributes.

ĐẠI HỌC DUY TÂN
International School
Khoa Đào tạo Quốc tế
http://kdtqt.duytan.edu.vn

What Views are Available?

- Plenty! Too many!
- Already, we've seen 15 different views, and many more are available.
- An architect needs a way to choose the useful ones.
- One thing that would help is to organize the views into broad categories.

What Views are Available? - 2

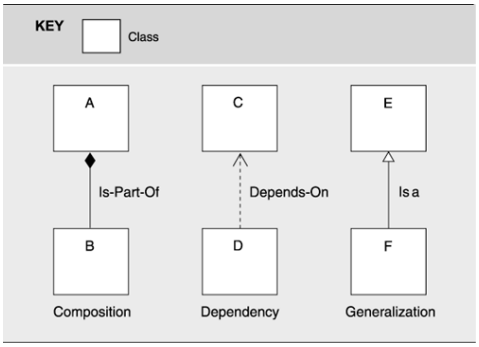
- An architect must consider the software from at least three perspectives:
  - How is it structured as a set of code units?
  - How is it structured as a set of elements that have run-time behavior and interactions?
  - How does it relate to non-software structures in its environment?
- Each of the views described earlier fall into one of these three categories, which we call *viewtypes.*

Exercise

- Name some view-based documentation that you like the most and explain why.
- Can we remove 1 principle of documentation?

The Module Viewtype – 1

- How is it structured as a set of code units?
  - Module *viewtype* - shows elements that are units of implementation.



The Module Viewtype – 2

- How is it structured as a set of elements that have run-time behavior and interactions?
  - Component-and-connector *viewtype* –shows elements that have run-time behavior and interaction.
- How does it relate to non-software structures in its environment?
  - Allocation *viewtype* - how how software structures are allocated to non-software structures.

The Module Viewtype – 3

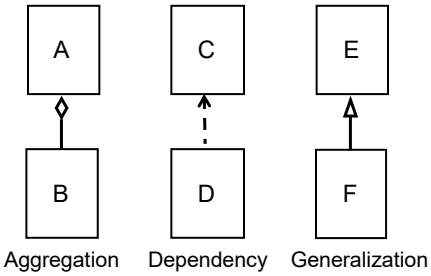
- Elements: Modules. A module is a code unit that implements a set of responsibilities.
- Relations: Relations among modules include
  - A is part of B. This defines a part-whole relation among modules.
  - A depends on B. This defines a dependency relation among modules.
  - A is a B. This defines specialization and generalization relations among modules.
- Properties: name, responsibilities, and visibility of the module, and its interface

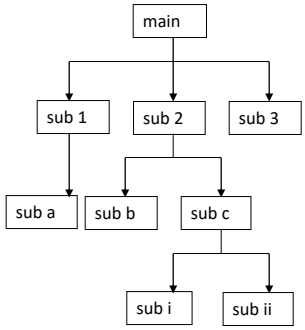
Module ViewType Uses

- Construction - These are the blueprints for the code. Modules are assigned to teams for implementation. Modules are often the unit for subsequent design (e.g., of interfaces).
- Analysis - Traceability and impact analysis rely on implementation units. Project management, budgeting, planning, and tracking often use modules.
- Education - A new developer will learn the system's structure by looking at module views.

Common Notations for Module Styles

- Informal: box-and-line
- Structure Decomposition
- UML: Class diagrams





Component-and-Connector (C&C) Viewtype

- Elements:
  - Components: principal units of run-time interaction and data stores
  - Connectors: interaction mechanisms
- Relations: Attachments of components' to connectors'
- Properties:
  - name
  - functional responsibilities
  - quality attribute volumetrics: how much, how fast, how many, how often, and so forth

C&C Viewtype Uses

- A starting point for the architect to show how the system works
  - Support reasoning about run-time quality attributes such as performance, security, availability,…
  - To answer questions such as:
    - What are the key components and how do they interact?
    - What are the key shared data repositories?
    - Which parts of the system are replicated?
    - How does data move through the system?
    - What parts of the system run in parallel?
    - How can the system's structure change as it executes?

Common Notations for C&C Styles

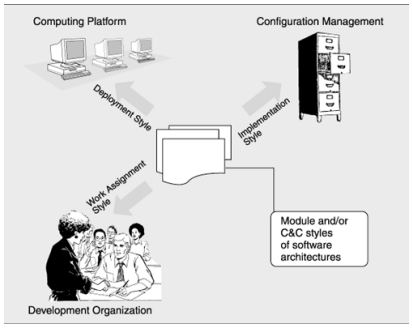
- Informal:
  - Box-and-line diagrams.
  - Most box-and-line diagrams showing run-time behavior are in fact attempting to be C&C views.
- Formal
  - Architecture description languages: Acme, Wright, UniCon, MetaH, Rapide, …
  - UML
    - Sequence diagrams, but not a straightforward mapping and often weak (i.e. not descriptive enough)

The Allocation Viewtype

- Elements:
  - software elements usually as defined in module or C&C viewtypes
  - physical elements from the operational environment
- Relations: varies, but often includes “allocated to”, “connected to”
- Properties: Various, according to style.

Allocation Viewtype Uses

- Deployment
  - Allocates software elements to processing and communication nodes.
  - Describes system infrastructure and configuration.



Allocation Viewtype Uses

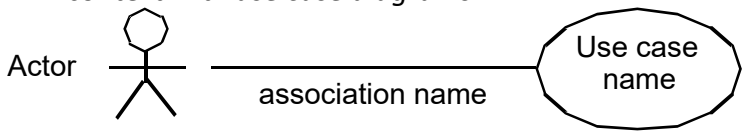
- Implementation
  - Allocates software elements to structures in the development environment's file systems.
  - Allocates elements from one view to those of another view.
  - Configuration management.
- Work assignment
  - Allocates software elements to organizational units.

Context Diagrams

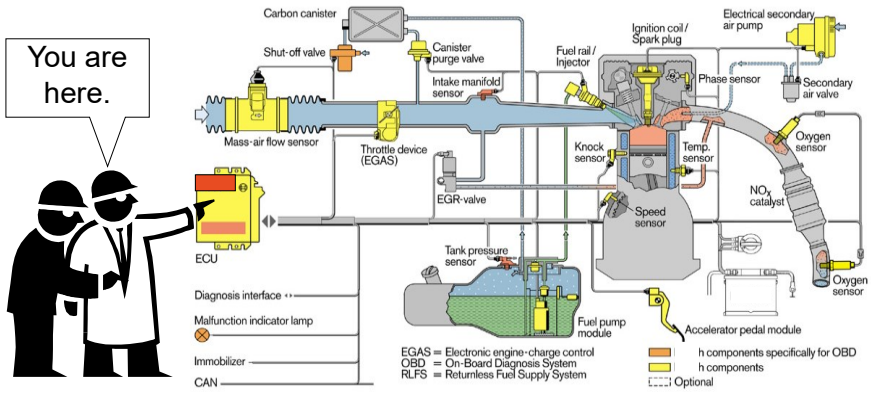
- Since view packets may show different parts of a system, and at different depth, we need a way to tell a reader where we are.
- This also holds true for view packets whose scope is the entire system.
- A *context diagram* fills this need.
- The primary function of a context diagram is to show what's *in* and what's *out* of the view packet.

Common Notations for Context Diagrams

- Informal
  - Box-and-line drawings with the system (or part of it being defined) clearly identified.
  - Tables listing external entities and the interactions with them.
- Formal
  - UML - No explicit context diagram. UML implicitly shows context with *use case diagrams*.



Example Context Diagram



Documenting Interfaces

- An interface is a boundary across which two independent entities meet and interact or communicate with each other.
- An *interface specification* is a statement of elementary properties that its architect chooses to make known.
- In most cases, this is where the architect's design ends and downstream, detailed designers go to work.

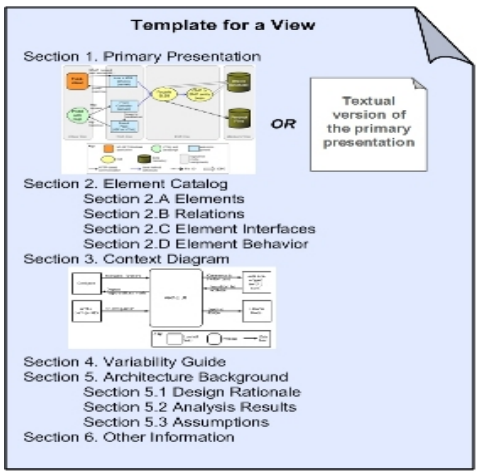
Guidance for Documenting Interfaces – 1

- Identity of the interface.
- Identify the resources provided.
  - The signature or syntax of the resource: information needed to use it in a syntactically correct manner.
  - Semantics of the resource: what is the observable effect of using the resource?
    - assignment of values to data or variables
    - changes in state or other resources
    - events signaled, messages sent, or events observable in the system's environment
  - Usage restrictions on the resource.

Documenting Interfaces - 3

- Identify locally-defined data types.
- Identify exceptions that can be raised by the resources on the interface.
- Describe any variability provided by the element's interface and how to exercise it.
- Describe the quality attribute characteristics.
- Indicate what services/date the element requires.
- Provide implementation notes and/or a usage guide.

Documenting A View



Guidelines for Documenting a View – 1

- Create a context diagram.
  - keep it simple – intuitive
  - shows how the system (or the portion being depicted in this view, view packet, or document) relates to its environment
- Create the primary representation or view.
  - describe the perspective (the viewtype)
  - usually graphical, but doesn't have to be
  - create a legend - *I like to create the legend BEFORE I create the picture. This forces you to be consistent!*

Guidelines for Documenting a View – 2

- Create an element catalog that explains the elements depicted.
  - tables work great for this
  - some information to consider including in this catalog include:
    - element names and brief description of responsibilities, interfaces, behavioral/quality attribute properties, relations, and any exceptions or additions to the relations shown in the view
    - organize by view, view packet, or document

Guidelines for Documenting a View – 3

- Create a variability guide if applicable.
  - describe those mechanisms used for achieving architectural variability and when they can be exercised, such as how to configure and/or instantiate an element or system
- Describe and/or document relevant interfaces.
  - if the view, view packet, or document describes elements that have interfaces they should be described in detail or reference the appropriate documentation
  - if referencing separate interface documentation, tables are handy

Guidelines for Documenting a View – 4

- Document the thought process, including
  - The rationale for design decisions that apply, along with key rejected alternatives and factors that influenced the design decisions.
  - Any analysis data and/or the results of experiments used to validate the design decisions.
  - Any assumptions made.
- Other information might include
  - system-specific and project-specific information
  - configuration management and ownership information
  - mapping to architectural drivers

Outline

- Role of documentation in the process
- Qualities of good documentation
- View based documentation
- View organization
- Designing the documentation

Organizing Views - 1

- Typically stakeholders require information across views
- Think about modifiability – in order to understand the impact of a change you may need information from:
  - "Uses" view
  - Generalization view
  - Data model
  - Perhaps even some dynamic and allocation perspectives

Organizing Views - 2

- There are a few of ways to organize the views
  - Provide a roadmap that describes how to navigate the views
  - Create a "packet" containing relevant snippets from the required views
  - Explicitly map between views

Providing a Roadmap - 1

- Recall that you are writing this document from the reader's perspective (hopefully)
- To facilitate the intended use of the document it is helpful to have a section describing how to use the document
- This section should contain
  - The intended uses of the document
  - Directions for how to obtain the information needed to support the intent

Providing a Roadmap - 2

- The roadmap is often hierarchical
  - You have a high level overview that points to the required sections
  - In those sections you have more detailed instructions
- This underscores the need to have standard organization

Creating Stakeholder "Packets"

- Another option is to have "packets" or sub-documents for a specific purpose
- These documents cut across views to provide the required information
- This "packet" will contain only the portion of the views needed to support the intended purpose
- It will also contain instructions for how to read the document (as in the roadmap)
- The issue with this approach is the initial effort and configuration management of the document
  - Typically requires more robust tool support

Mapping Between Views - 1

- There are two option for mapping information from one view to another
  - Build a “bridge” document that relates elements form one view to elements of another
  - Create a “combined” or “hybrid” view

Mapping Between Views - 2

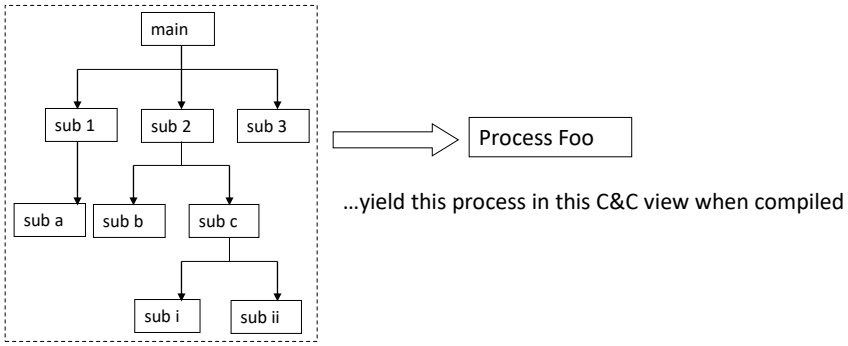
- Creating a combined view can be dangerous
  - We’ve talked about maintaining a consistent perspective
  - The mixing of perspectives can “muddy” your thinking
    - And thus any analysis you do is suspect
- You must be very clear about what you are conveying
  - And ensure that even your hybrid view maps to things in the real world

Hybrid Views

- One common "hybrid" view is a mapping between modules and processes (or threads)
- You can show what happens to code when it is compiled and executes
  - This is important for analysis that involves both static and dynamic perspectives
- Overlays can be an effective way to do this
  - Where you "overlay" one view with another

Hybrid Example

This elements of this module view…



…yield this process in this C&C view when compiled

Outline

- Role of documentation in the process
- Qualities of good documentation
- View based documentation
- View organization
- Designing the documentation

What Views are Needed?

- Build a document where
  - the ROWS enumerate the stakeholders.
  - COLUMNS enumerate the set of documents (views, viewpackets, or complete documents) that *could* apply to the architecture being documented.
  - Check box (x,y) if stakeholder x would like view y.
- Combine documents where practical and appropriate appropriately to reduce their number and overlap.
- Prioritize documents based on schedule and the needs of stakeholders.

Example Documentation Organization Table

Stakeholder	Document 1	Document 2	Document 3	Document 4
Project Manager	1	1		1
Member of Development Team	2	2	2	2
Tester and/or Integrator		2	2	
Maintainer		2	2	2
Product Line Application Builder		1	1	0
Customer				
End User				
Analyst	2	2	1	2
Infrastructure Support Staff	1	1		1
New Stakeholder	*	*	*	*
Current and Future Architect	2	2	2	2

KEY:

2 = detailed information

1 = some detail

0 = overview only

* = unknown

Note that "document" in the column might be a single view, combined views, a view packet, or a complete document.

Overall Documentation Guidelines – 1

- Describe the architectural drivers
- Provide a reader's roadmap
  - explains how the documentation is organized helps the stakeholder navigate through
  - list of views and a statement of what each view is for
  - lists scenarios for using the documentation, showing which sections should be consulted
- Provide a view template that explains how each view is documented and illustrates the *standard organization* for each view

Overall Documentation Guidelines – 2

- Provide a system overview that includes.
  - an informal, prose description of the system, and its purpose and functionality
  - a context drawing (or drawings)
- Provide a mapping between views as necessary.
  - describe how the various views are related
  - tables are an excellent mechanism for clearly and concisely listing these relationships

Overall Documentation Guidelines – 3

- Provide a directory showing where all the systems elements and relations are defined and used.
- Provide an architecture glossary and acronym list.
- Provide the background, design constraints, and rationale that is not provided in the views.

Group Work

- Draw Module View for any online shopping system that you like.

References - 1

- http://www.sei.cmu.edu/architecture/arch_doc.html
- https://wiki.sei.cmu.edu/sad/index.php/The_Adventure_Builder_SAD
- Clements P.; Bachmann F.; Bass L.; Garlan G.; Ivers J.; Little R.; Nord R.; Stafford J.; *Documenting Software Architectures: Views and Beyond,* Reading, MA: Addison-Wesley, 2002
- http://www.sei.cmu.edu/architecture/presentations.html

References - 2

- https://www.youtube.com/watch?v=6QsSuQPxunk (Software Design Document)